Sécurité des logiciels

NX? Not on the stack? Still pwnd!

Samuel Thibault <<u>samuel.thibault@u-bordeaux.fr</u>> CC-BY-NC-SA

Stack overflow exploit needs...

Stack overflow exploit needs

- Executable stack
- Buffer on the stack
- Known position



Stack overflow exploit needs...

Stack overflow exploit needs

- Executable stack
- Buffer on the stack
- Known position



How to execute stuff without being executable?

Remember what our shell code was: Basically execve("/bin/sh", {"/bin/sh", NULL}, {NULL});

Do we really need to write code for this?









How to execute stuff			
without being executable?	0x1000	OvOfb8	
	0x0ffc		
Overflow exactly like this!	0x0ff8	0x0fb4	
Overnow exactly like this:	Nv∩ff4	0x0fac	
Det inte libe beek	UAULLA		
Rel-INIO-IIDC NACK	0x0ff0		
	0x0fec	execve	
What if we want to do more than just one	0x0fe8		
system call?	0x0fe4		
	•••	0x0fac ◄	
		→ /bin/sh →	
	UXUIAC		1

. . .

. . .

How to execute stuff			
without being executable?	0x1000	OvOfbQ	
	0x0ffc	UXUIDO	
Overflow exactly like this!	0x0ff8	0x0fb4	
eveniew exactly like this:	0x0ff4	0x0fac	
Det inte like heeld	UXUITA		
Ret-Into-IIDC hack	0x0ff0		
	0x0fec	execve	
What if we want to do more than just one	0x0fe8		
system call?	0x0fe4		
	• • •		
		→ /bin/sh →	
	0x0fac		I I

. . .

. .

How to	execute stuff
without	being executable?

A second chance...

First "return to" foo



How to execute stuff without being executable?

A second chance...

First "return to" foo Now in foo, then "return to" bar



How to execute stuff		
without being executable?	0x1000	
0	$0 \times 0 \text{ ffc}$	
A second chance	0x0ff8	
	ESP _{x0ff4}	
First "return to" foo	0x0ff0	bar
Now in foo, then "return to" bar	0x0fec	foo
Now in bar	0x0fe8	
Err but then har's peremeters pood to	0x0fe4	
EII, but then bars parameters need to		
be almost the same as foo's	•••	
But bar does not have to be a	0x0fac	
proper function!		• • •
proper function:		

How to execute stuff	0v1000	
without being executable?	0x1000 0x0ffc	
SP lifting hack	0x0ff8	
	0x0ff4	
A lot of glibc functions end with cleaning	0x0ff0	
the stack, e.g.:	0x0fec	
	ESP 0x0fe8	
bar:	0x0fe4	bar
addl \$12,%esp		foo
ret	•••	
	0x0fac	

. . .

How to execute stuff	0 1 0 0 0	
without being executable?	0x1000 0x0ffc	
SP lifting back	0x0ff8	
	0x0ff4	
A lot of glibc functions end with cleaning	0x0ff0	
the stack, e.g.:	0x0fec	
	ESP 0x0fe8	
bar: addl \$12 % arm	0x0fe4	bar
addi şiz, desp		foo
Tec	•••	
	0x0fac	••••
		• • •

How to execute stuff	01000	
without being executable?	0x1000 0x0ffc	
SP lifting hack	0x0ff8	
	ESP 0x0ff4	
A lot of glibc functions end with cleaning	0x0ff0	
	0x0fec	
bar:	0x0fe4	bar
addl \$12,%esp		foo
TEC	•••	
	0x0fac	•••

How to execute stuff	0	
without being executable?	0x1000 0x0ffc	
SP lifting hack	ESP 0x0ff8	
	0x0ff4	
A lot of glibc functions end with cleaning	0x0ff0	
the stack, e.g.:	0x0fec	
	0x0fe8	
bar:	0x0fe4	bar
addi \$12, %esp		foo
IEC	•••	
	0x0fac	

• • •

How to execute stuff without being executable?
SP lifting hack, complete story:
First ret-into-libc



How to execute stuff without being executable? SP lifting hack, complete story: First ret-into-libc Now in creat, ..., returns to sp_lift_12



How to execute stuff without being executable? SP lifting hack, complete story: First ret-into-libc Now in creat, ..., returns to sp_lift_12 Now in sp_lift, cleans stack



How to execute stuff without being executable? SP lifting hack, complete story: First ret-into-libc Now in creat, ..., returns to sp_lift_12 Now in sp_lift, cleans stack Stack cleaned, returns to exit



How to execute stuff without being executable? SP lifting hack, complete story: First ret-into-libc Now in creat, ..., returns to sp lift 12 Now in sp lift, cleans stack Stack cleaned, returns to exit Now in exit, exits nicely pwnd!



- How to execute stuff without being executable?
- SP lifting hack
 - Call chain can be arbitrarily long
 - Just need to find in libc what you need



What is the basic problem?

We can overflow

- from a data buffer
- into a control area



Stack overflow exploit needs...

Stack overflow exploit needs

- Executable stack
- Buffer on the stack heap
- Known position



```
int litentier(void) {
  int i;
  char buf[64];
  printf("> ");
  fflush(stdout);
  gets(buf);
  i=atoi(buf);
  return i;
```



```
int litentier(void) {
  int i;
  char *buf = malloc(64);
  printf("> ");
  fflush(stdout);
  gets(buf);
  i=atoi(buf);
  free(buf);
  return i;
}
No way to overflow into the stack
Something else?
```

	Stack	rw–/x
	Libraries	
	mmaps	
	Неар	rw–
	Bss	rw–
	Data	rw–
	R/O Data	r—
	Text	r–x
 0		

Heap structure (simplified)

When you call **malloc** three times



data3

	datau	
Heap structure (simplified)		
When you call malloc three times	data1	
There are actually some additional headers		
	data2	V
	data3	
		•
	data4	28

Heap structure (simplified)

When you call **malloc** three times

There are actually some additional headers Which allow to jump between allocations:

- BK (back)
- FD (forward)



Heap structure (simplified)

When you call **malloc** three times

There are actually some additional headers Which allow to jump between allocations:

- BK (back)
- FD (forward)

Now free(data2)

Have to update BK/FD





Heap structure (simplified)

When you call **malloc** three times

There are actually some additional headers Which allow to jump between allocations:

- BK (back)
- FD (forward)

Now free(data2)

 Have to update BK/FD BK = data2->BK; FD = data2->FD; BK->FD = FD; FD->BK = BK;





So basically free(data2) does

- BK = data2->BK;
- FD = data2->FD;
- BK->FD = FD;
- FD->BK = BK;

Now, what if I overflow data1?

- I can choose data2's BK at will
- I can choose data2's FD at will



So basically free(data2) does

- BK = data2->BK;
- FD = data2->FD;
- **BK->FD = FD**;
- FD->BK = BK;

Now, what if I overflow data1?

- I can choose data2's BK at will
- I can choose data2's FD at will



So basically free(data2) does

- BK = data2->BK;
- FD = data2->FD;
- **BK->FD = FD**;
- **FD->BK = BK**;

Now, what if I overflow data1?

- I can choose data2's BK at will
- I can choose data2's FD at will

In the end, I can divert the **control** in the stack



Countermeasures

Libc checks for coherency of headers

• More expensive free

Static code analysis / compiler-provided information

• Check for array bounds

Stack overflow exploit needs...

Stack overflow exploit needs

- Executable stack
- Buffer on the stack
- Known position
 - Next time!

